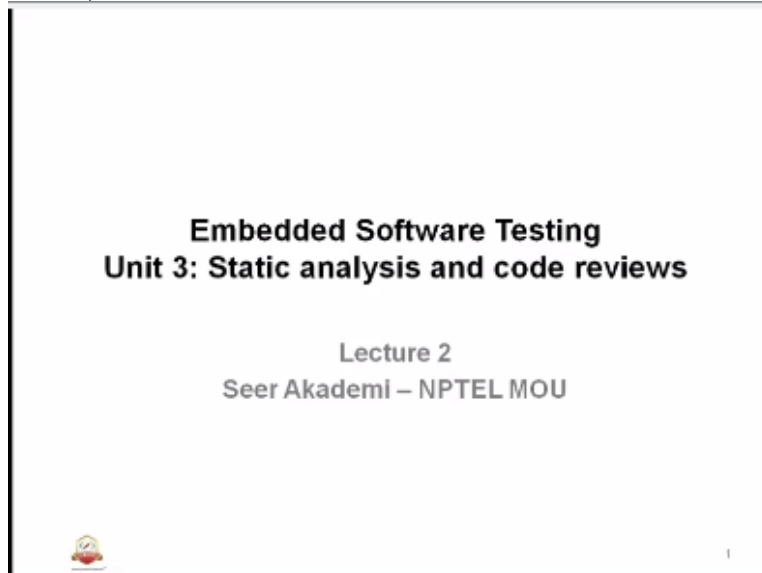



(Refer Slide Time: 00:04)



Embedded Software Testing
Unit 3: Static analysis and code reviews

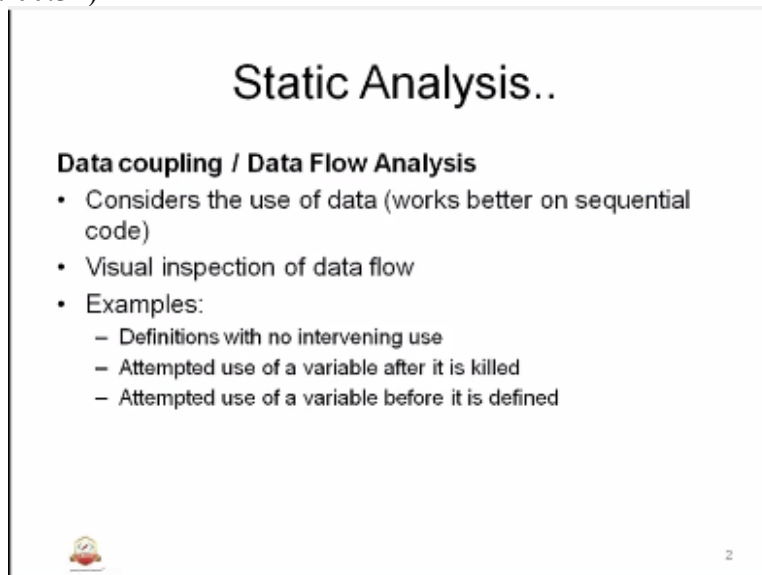
Lecture 2
Seer Akademi – NPTEL MOU



1

Welcome to the next session of embedded software testing unit 3 this second session of this unit 3 and we will study about static testing aspects like static analysis code reviews and today we will start on the.


(Refer Slide Time: 00:32)



Static Analysis..

Data coupling / Data Flow Analysis

- Considers the use of data (works better on sequential code)
- Visual inspection of data flow
- Examples:
 - Definitions with no intervening use
 - Attempted use of a variable after it is killed
 - Attempted use of a variable before it is defined



2

Data coupling and data flow analysis, before that we will just have a recap of what we studied in the earlier session we had look in to.

(Refer Slide Time: 00:45)

Static Testing

- *A process of evaluating a system or component based on its behavior without executing the program.*
- *Context with Dynamic testing: A process of evaluating a system or component based on its behavior during execution*



Static testing and we know that it is a component based testing without execution of the embedded software program we will do a evaluation of a the system without the need of executing the program so by analyzing the various aspects of the embedded software aspects in contest link the dynamic testing statistic in the execution of the embed.
(Refer Slide Time: 01:20)

Static vs. Dynamic testing

- Two techniques complement each other
- Static analysis were explicitly used in olden days embedded software testing where tools availability and affordability were on stake.



So basically for completion of embed software testing, we need to have both this type of techniques to be tested other complimented, these two techniques for each other completion
(Refer Slide Time: 01:38)

Static Testing ..

- Static analysis
- Reviews, inspection and the test process
- Testing Metrics



6

So the different types of static testing is static analysis, reviews, inspection and the test process the we have testing metrics we can use for reporting, tracking and make sure that the testing is complete in all aspects.

(Refer Slide Time: 01:56)

Static Analysis

Static Analysis

“Analysis of a program carried out without executing the program” – BS 7925-1

- Unreachable code
- Parameter type mismatches
- Possible array bound violations
- Faults found by compilers
- Program complexity (as the complexity increases, the fault density increases)



7

In static analysis we have gone through the type of analysis that we do in unreachable code, parameter type mismatches, possible array bound violations, faults that are found by the compilers it could be general info type warnings or any errors. So these type of issues with the reports with the compilers are also analyzed static, and with the help of that we will lead program complexity as the complexity increases, the fault density increases.

We know that we should limit the complexity to certain extents so that the complexities of the program will control and it will be in the limit.

(Refer Slide Time: 02:44)

Static analysis..

- Static analysis can be used as soon as the code can be compiled
- They can find bugs early in the development cycle, and bugs found earlier are less expensive to fix.
- They do not require program inputs, so bugs can be found and eliminated without incurring the expense of developing test cases.
- They can make it easier and less expensive to develop dynamic test cases. The consequence is they can eliminate more bugs for less expense.



www.embedded.com

8

Also we had highlighted about the advantages of this static analysis when it can be done as soon as the code is done and compiled it can be used as it is, we have the understanding of the program and requirements and on progress that whether it is there or not so you can do a pre-Noreen sort of a earlier of the development of the program or the code so that is why static analysis is effective as well as the code is done better to do it in the beginning then as in the end or the after dynamic testing is completed.

(Refer Slide Time: 03:29)

Static Analysis..

- % of the source code changed
- Graphical representation of code properties:
 - Control flow graph
 - Call tree
 - Sequence diagrams
 - Class diagrams



9

Also we have a representation of the control flow, call tree, sequence diagrams, class diagrams of the embedded software system so that is also used for analyzing the project or the program so that also will be part of static analysis.

(Refer Slide Time: 03:49)

Static Analysis..

- **Control Coupling and Data Coupling**
- Definition from Cast-19 paper from FAA
- *“Data coupling - The dependence of a software component on data not exclusively under the control of that software component.”*
- *“Control coupling - The manner or degree by which one software component influences the execution of another software component.”*



10

Static analysis so the main aspect is control coupling, data coupling. Data coupling, the dependence of a software component on data not exclusively under the control of that software component that means the dependency of the component on particular data is not getting controlled by the same component getting used or saved between the different component all the data is been coupled between this component with or the data component.

And control coupling, the manner or degree by which one software component influences the execution of another software component so that is what the control coupling is about the basically we do an analysis for the various parts of the program in terms of how is the control is being done in the entire life of the software programming embedded software program is been alleged that is what we do with the control coupling.

So based on this definition from the cast paper it is from the FAA federal aviation academy, it is one of the mandatory group which qualifies or certifies the evidences of the airlines or aero space's software products. So example for so we have studied in the earlier section.

(Refer Slide Time: 05:25)

Static Analysis..

Control Coupling / Control Flow Analysis

- Considers the use of program and its flow
- Visual inspection of control flow
- Examples:
 - Call tree unreachable completely or reaches partially
 - Parametric issues



12

And with the help of control coupling, control flow analysis we can see there are unreachable part partially or completely and there are parametric issues sop that we flow is not proper and we can have a visual inspection of the control flow.
(Refer Slide Time: 05:48)

SW Complexity

Static Metrics

- McCabe's Cyclomatic complexity measure
- Lines of Code (LoC)
- Fan-out and Fan-in
- Nesting levels

Also we do a software complexity is called static metrics which we study in detail in this description, mCabe's cyclomatic software measure, lines of code, nesting levels, fan out and fan in.

(Refer Slide Time: 06:04)

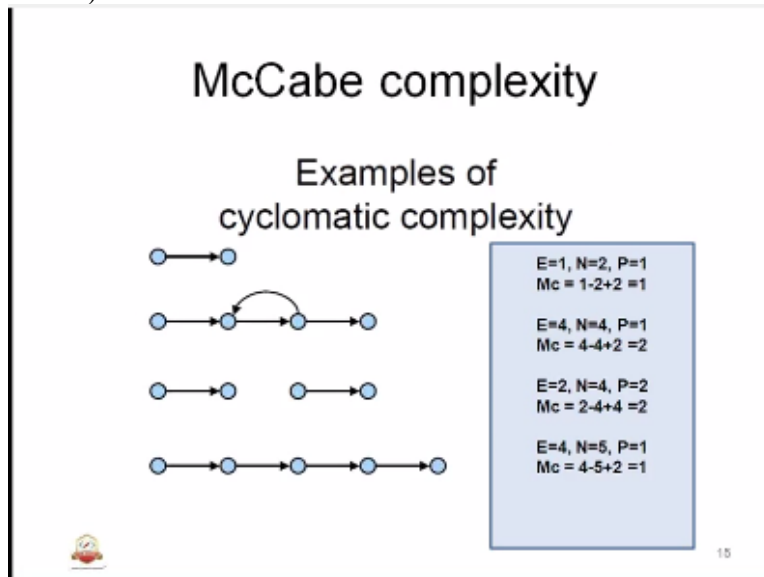
McCabe complexity

- Cyclomatic (McCabe) Complexity calculation:
- $C(g) = E - N + 2 *$
 - E : Edges
 - N : Nodes
- If $C(g)$ is > 10 meaning that the complexity is too high and calls for rework
- $C(g)$ specifies number of independent paths the program has.
- * in case of disconnected notes it will become $2P$

Then we had gone through cyclomatic McCabe complexity with details like cyclomatic complexity calculations of edge –nodes+2, and if in case of disconnected parts are there it will become P depending on the number of part, so complexity basically dependant on the edges and the nodes so these are derived out of its diamonds or the closing blocks within the software program.

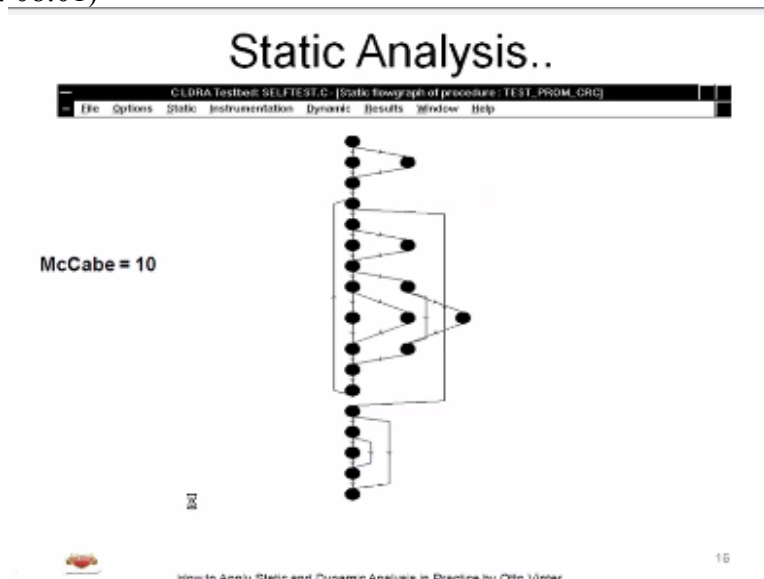
So if the software complexity > 10 it means the complexity is high and some of the industries they call for the particular product so because you won't have a complex program because the chambers of errors are very high, so basically the complexity number specifies the number of independent paths the program has, and we also saw the examples of the cyclomatic complexity very important part.

(Refer Slide Time: 07:10)



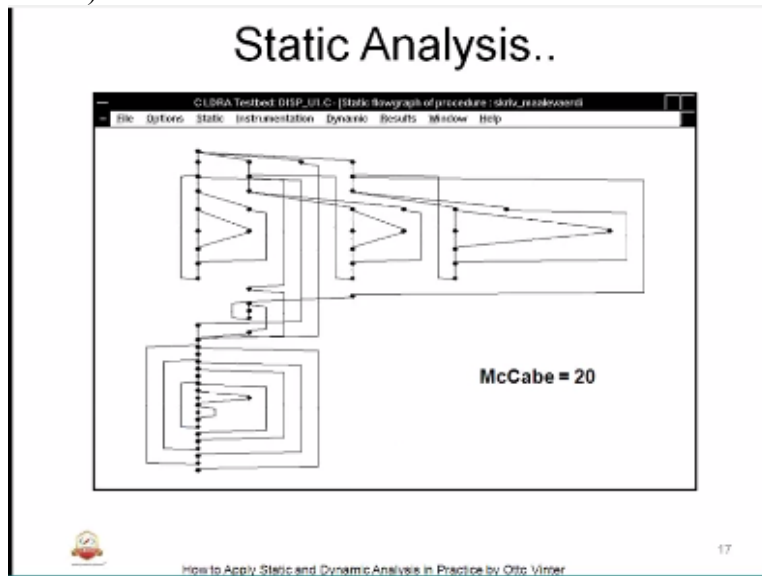
In the case we have only one the complexity is one because we have two nodes and one edge, In the second one we have four nodes and four edges so here the complexity is two and next one we have the complexity is two why because there are two paths and they are disconnected with the help of four nodes and there are two edges and in the last one we have five nodes and four edges when you say that it will become one complexity is one we can see the independent execution path is only one from the first node to the last node, and some more examples reported from.

(Refer Slide Time: 08:01)



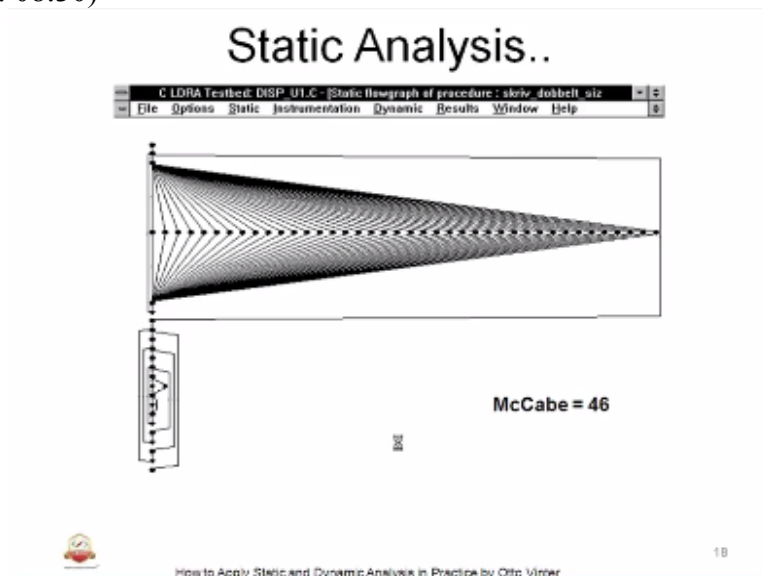
The tool such understand c++ and maybe you can go through that this is from the LDR related because you can see the various parts being anode with different nodes and the edges. Here we method is 10.

(Refer Slide Time: 08:20)



In the next on we have more complexity there are lot of nodes having connected with different edges the complexity here it is 20.

(Refer Slide Time: 08:30)



In the next on we have complexity as 46 where we have definitely we call for because the program can crash or program may have lot of errors and bugs and it is difficult to fix so this kind of a complexity get avoided so with the help of tools it is easier to analyze such complexity so that hence the tools are getting used.

(Refer Slide Time: 09:03)

Static Analysis..

Data coupling / Data Flow Analysis

- Undefined but referenced variables
 - must be removed from the code
- Variables defined but not used in scope
 - should be documented in the code
- Variables redefined with no use in between
 - should be documented, if not close to

- Suspicious casting (loss of information, mismatches)
 - if unavoidable, use explicit casts
 - should be documented

- Global variable anomalies (local overrides global)
 - should be removed



And static analysis we will study we will continue today we know that the control coupling are the throughout the program controlled and we need independent execution path as we analyzed, so in data coupling basically it depending on objects of the data which is been used entirely in the software program.

So data will work better on sequential code do you know that? When there is a complex code so you need to analyze the data in terms of it seems stage at the flow.

So ideally we will do a visual inspection of the data flow examples we will go through the definition with no intervening use of this we should just want to understand that what is your depth link usage? And attempted use of a variable after it is killed that is specially linked in the embedded surfaces the how the data is being used? How the data is being initialized in that program? And like while attempted use of a variable of it is killed.

Similarly after defined how the attempted use of variable use in the entire embedded software program, so that also will be analyzed in the data coupling or the data flow analysis.

(Refer Slide Time: 10:54)

Static Analysis..

Data coupling / Data Flow Analysis

- Undefined but referenced variables
 - must be removed from the code
- Variables defined but not used in scope
 - should be documented in the code
- Variables redefined with no use in between
 - should be documented, if not close to

- Suspicious casting (loss of information, mismatches)
 - if unavoidable, use explicit casts
 - should be documented

- Global variable anomalies (local overrides global)
 - should be removed



Coming to the next slide static analysis, data coupling, data flow analysis so there are data which are undefined but referenced variables we have so we should remove from the code, on this early we have some data which are either referenced but undefined so things of point in either similarly we have variables defined that not used in this scopes either it should be documented in the code or it should be mentioned separately.

And variables redefined with no use in between should be documented if not close out project if you don't want to have are definition of the without usage and we should document it documentation is you should comment it if it is going to be used for latest thing or if not used you can close that variable and leave it. Then you know that costing, costing is very important in the embedded software like the different types of variables,

Being interchanging the contents of that so while doing that we use the type cast and all in that with the help of embedded C and while doing that with the help of that we lose the information in terms of son gates being discarded and also this is the charger there is a mismatch between the internal use and the actual use, so if unavoidable use explicit casts better you explicitly type cast like particular variable.

So we have variable temperature assigned to variable read Val is also read, suppose if this is a 16 bit integer and this one is a 32 bit integer then the rest would be temp should be aligned with the help of 32 bit type casting with the read Val so that so it will be aligned but we should be careful such a way that the actual value is that for 32 we will not lose any information it could be vise versa also if it is 16 bit that is be assigned with the help 32 bit.

On the right hand side transfer that the data outside the 16 bit remains so it need to be appropriately using it when we doing the castings such as it can analyzed in static analysis and there could be mismatch between the data what is been used? And what is been assigned? So that also will be brought out while doing the static analysis of the data flow and the data coupling. Global variable analysis local over rates and global so we know that we used global variables which are used a parameters shade of that between various function of the closing here and sometimes.

The same name are used between the local function and it is confused in terms of usage and while using that function assumes that we programmer code has written assumes that the local variable will be updated or the global variable will be used or updated with the function.

But the local gets the priority of the it over rates the global basically so such analysis should be removed the best way to have this avoid the redundant variables it should have a meaningful variable in the ends so that it is very clear it could be a local or it could be a local so these things definitely brought out while doing this static analysis saying that it is the anomaly between global versus local in terms of assignment or parameter passing.

So all this can be brought out doing the static analysis, typically some standards that is why the you start the global variables with global variable one this is like and local variable start with a variable so it will very clear by seeing variable itself you know that what type of variable and where and it is used? How it is getting used? Similarly local variable getting identified with the small letter variable so it is a clear segregation of data,

And there are no anomalies in terms of the usage and the flow, so this is how the data coupling can be unleashed, so next one is on being static analysis basically.

(Refer Slide Time: 17:08)

Static Analysis..

- **Static Analysis in Unit Testing**
 - Establish project standards for code
 - no goto's, breaks in all cases, procedure size, etc.
 - Remove unused items
 - unreachable code (incl. procedures)
 - declared variables that are not used
 - Address design architecture problems
 - use visual inspection of control flow graphs
 - procedure parameter anomalies (referenced only, defined only, not used)

How to Apply Static and Dynamic Analysis in Practice by Otto Winter

The use it in the unit testing why because unit testing is done on the implementation of the code basically that is why it is adorable to use during the unit testing when we do the data flow basically so I will bring the analysis of the code certain things like project standards for code we need to establish the coding standards and the Norman culture of the created ones and the procedure sides or the functional side calls and breaks in cases like.

How many cases it can be? Some of the DSPS is not allowed more than 4 cases, in that case we need to be strictly following the roles of that particular process or the development environment such that the program is not have any errors or the it will not go for a class, similarly we know that we should not have go to we know that not a good practice t have because it will have unintended program and it is very difficult to return somewhat.

So this things are part of the project standards that is to be established and analyzed against that so most of this will be done in the unit testing and data and the control flow can be equally done while doing the unit testing and another aspect could be removed unused items. That mean there are certain unreachable code to the function within the code of the variables and declare that whether there are sort of not all this parts can be identified.

Maybe the inspection, review or with the help of tools all these can be avoided so such as un reachable code declared variables that are not used so this can be avoided in the instrumentation, so such things will be brought out during the static analysis of the implemented program. And the other important aspect of the static analysis is address design architecture problems there are the flow control flow and the data flow.

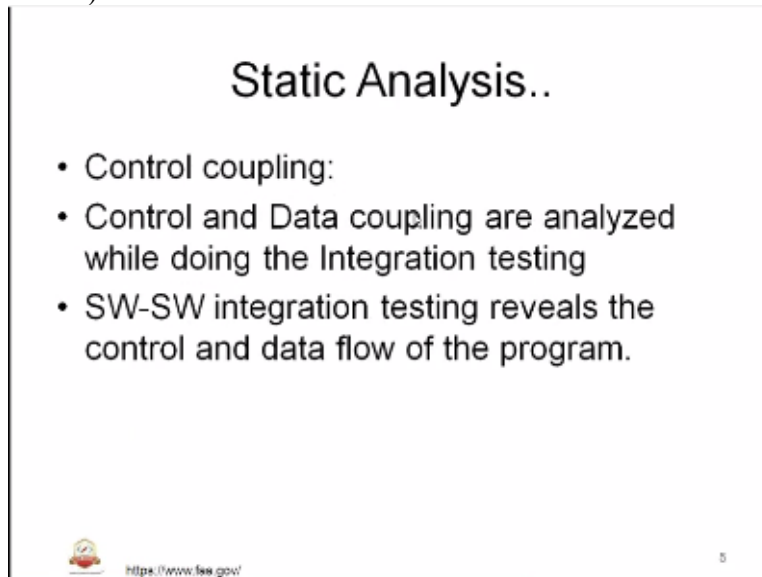
Typically program which is having clear statements the switching statements with the help of pointers or the addresses that will be difficult to find out while we do the execution whiter done for all inspection or typically inspection of the control flow so we are clear about the architecture or the program of the program is designed and the various states this is definitely useful in ends states or state machines etc.

So in this case better to have a understanding of the entire program then with the help of visual inspection apply the standards of the architectural rules or the architecture that we have

understood in the program and with the help of control flow we will analyze the fitting and we will understand the entire program similarly we have anomalies in the procedures, procedures are defined referenced and not used such procedures,


Which will avoid and the usage of procedures in terms of parametric while used also need to be analyzed while doing the static analysis that is also an important aspect. So coming into the next one,

(Refer Slide Time: 21:36)



Static Analysis..

- Control coupling:
- Control and Data coupling are analyzed while doing the Integration testing
- SW-SW integration testing reveals the control and data flow of the program.

 <https://www.fee.gov/> 5

We know that control coupling, control and data coupling are analyzed while doing the integration testing also it is not just enough to have control coupling and data coupling controlling, the unit testing will also important to have the control and static data coupling during the integration testing why it is important? In terms that while doing the integration testing we will address the different modules,

And we know that we are going to integrate various modules and the modules are with respect to software to software that mean application level I could be software to hardware such as device drivers could be application system versus application integration, such sort of a bottom up or the top down approach integration testing while doing that it is better to analyze the integrity details of the modules.

So that the control and data coupling can be analyzed, So that is the highlight of that control flow and data flow while doing the integration testing, the next one is being the software, software integration testing reveals the control and data flow of the program we know that basically the software is up to integration as I said application level or the program level between the modules of the entire program will definitely bring out the control flow of the program.

And the data and the objects and the variables that are flowed on within the program, so that is how the static analysis can be done in the entire embedded software program so that is how we will do the control coupling and data coupling with the static analysis.

(Refer Slide Time: 23:58)

Static analysis tools

- A static analysis tool is like an automatic reviewer for your code. It reads the source code (without executing it) and looks for cases where it will behave in an undesirable manner—for example, dereferencing a null pointer, dividing a number by zero, or overflowing a memory buffer.
- Static analysis tools do not depend on sample input—they can infer the software's behavior based on just the source code. When a bug is found, the tool reports its location to a software engineer, along with the information needed to diagnose the problem.
- Since they do not depend on sample input, static analysis tools can investigate program behavior in corner cases that are not anticipated by testers and human inspectors.
- While no tool can find all bugs, modern static analysis tools generate valuable results with minimal false positives, even for projects with millions of lines of code.



www.embedded.com

5

So now we will come to the tools part so what are the tools that are used? And how they are being used? And what way it is used to? First static analysis, static analysis tools is like a automatic reviewer for your code that means whatever the human being does the tool will take care of that basically on the independent embedded software program. So if basically content is the code or reads through the source code.

Of course without execution and looks for cases where it will behave in an undesirable manner for example dereferencing a null pointer, dividing a number by variable by zero or overflow of a memory buffer, we know that embedded software program should definitely have a upper bound and lower bound of an memory and id these variable are procedure in referring that so it will identify those elements.

So static analysis tool will look like a arithmetic reviewer of the code static analysis tools do not depend on sample input definitely no input dynamic input are or actual is required they can infer the software behavior based on the source code because source code will have definition and the flow of the variables that are intended to the used it can be procedure of the function so with the help or that so all the tools does it.

It will try to feed a various values and it will try to analyze the flow of the particular parameters based on its definition and the declaration, so that is what the meaning of second bullet like the infer the software behavior based on just the source code so when a bug is found the tool reports its location to the report or the report or the software engineer who ever is running the tool, as all with the information needed to there must be a problem.

That means it will just point out the problems where the issues builds there in terms of the static analysis aspects of the what we have seen earlier in the control flow data flow issues, which is automated which is done in the tool, and it is report the location basically it will identify the which location and what is there, so this is very important why because there are rule checker that are been used and that is to identify various anomalies within the implemented program especially the rules like mismatch it is a modern industry standard rules so missed out standards that was 2004 has 120 plus rules are there.

Those are rules which are still in to be followed to the implemented in the program, so they are mandatory rules and guide lines that also will be understood by the tool where the violation is there, violation of the rules and indented usage of the program in to data it will report the error so since they do not depend on sample input static analysis tools can move to program behavior in corner cases that are not anticipated by testers and human inspectors.

Some of them such as very small sort of a like fragment or if it is a macro usage which is very difficult to analyze or inspect visually so those things are definitely can be caught out with the help of the static analysis tools because for that the program or dependency or the a sample input is not requires will simple infer in the program flow or it has return in the data flow of the embedded software tool will identify such issues.

Which are hiding from the human while during the inspection, while no tool can find all bugs of course modern static tools will generate a valuable results with a minimal false positives at least it will definitely aid the tester of the testing team in terms of where could be the problem areas where are the issues whether the correctly that is going to happen while the program is going to go for a fed or program getting executed.

So even with projects with millions of lines of code definitely the tools will give some sort of a hint or issues or it can report directly but where are the problems which it cannot report directly or it errors or issues so because of one issue it could result in a run time error suppose it identifies a variable as a variable improperly defined in each issue this at least will give a issue suppose you say issue done this issue.


When these definite a clue for the program that can have a bug while executing on the field or then there is a dynamic message of the program it could be any issues like memory, performance, speed, timing etc anything it could be so at least if it is not in integrity between the support but it can do it can aid or help the tester in terms of results that can have a some sort of a positive news in terms of errors or issues.

That can be fixed to have a major crack down in the latest model of the program so that is how it can be used the tools can be used. So what are the tools that are available in the market?

(Refer Slide Time: 31:11)

Static analysis tools

- Understand for C/C++ or Ada from Scitools
- Polyspace, Coverity, QAC, Cantata, LDRA testbed..
- MISRA tools inbuilt with IDEs (Multi, CCS..)
- Logiscope Rulechecker, PC-lint

 www.embedded.com 6

Or there are most likely to be used in the embedded software industry of course there are 100 of tools each one have been used or each ones have been reward based on the usage and the quick back from the embedded industry so there are certain few examples which I try to come through now which can be used directly or indirectly partially fully depending on the type of embedded software systems it can be any form fully available systems.

It could be a automotive or a set of space etc, one fetch tool is understand for C/C++ or Ada from scitools basically it is a good static analysis tool which will identify an static analysis aspects like McCabe complexity likewise of code it is report the dead code, dead objects, initialization variable in proper usage then the call tree you know all these by now so as a static analysis aspect all this can be done with the help of understanding process C/C++ sort of a features.

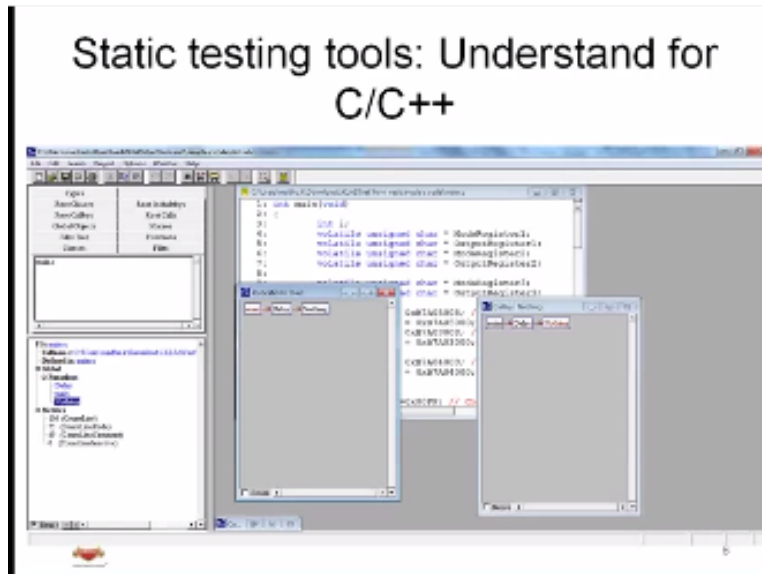
I will try to go through the static analysis of the server optical of the understanding of the tool in the next slide probably similarly we have a tool like all the space we can have advantages and disadvantages they will use it in the automotive industry similarly we have quality for inspection and assumes and static analysis we have a QA checker QAC, cantata, then we have LDRA test bed so some of these tools like LDRA or RDRT.

They can also be used for unit testing or for instrumentation you know that during white box testing we can do instrumentation of the code in the test drivers so need while this testing can be used for all these also can help in terms of static analysis, code instruction relives etc. of course I have said the missed tools the tool for a guidelines and rule checker we have the ideas which have code composed to do might be etc.

We have a inbuilt feature so which will which can be created and used on code so which will report a error of the violation of the missed out rule so in such rules have been violated and that violations can be reported as a static analysis output that is how this tools can be used, similarly we have logic scope, rule checker from logic this also must be copied into the tech cross industries they use it and we have PC lint.

So that is also one of the static analysis tool it is more on the olden days tool basically like as the systems they had and first program and inbuilt along with the compiler needs programs is all level it is a open source static analysis tool this can be used for static analysis so these aver some of the static analysis tolls so we will try to understand the short of static analysis testing tool. We have to understand for some purpose.

(Refer Slide Time: 35:29)



I think I will play to create a sample project and explain it the row or I will explain first purpose how we can use in one of the practical session we will see later part of the course probably in the embed software testing course, so this is the example software testing reports you can see this tool has lot of features such as we can build the program, build the project with the help of the embedded machine you can see source code here like INT main is there.

There is a blue letter which is understands from C++ build that declaration and differences that part implies the source it will highlight and the variable name it has accurate here so the definition can be understood by the users in terms of analyzing the code and understanding the source code and here you can see a invitation of a sample program which I have wrote there is a main delay which is called by main.

And delay is calling the another program called methane so this is called the call tree, so in this you can have complexity in terms of the one more slide I have it to show you at what deep level we can use the understand passing purpose and another window we can have another type of report the which shows the other side like the nothing is called by whom.

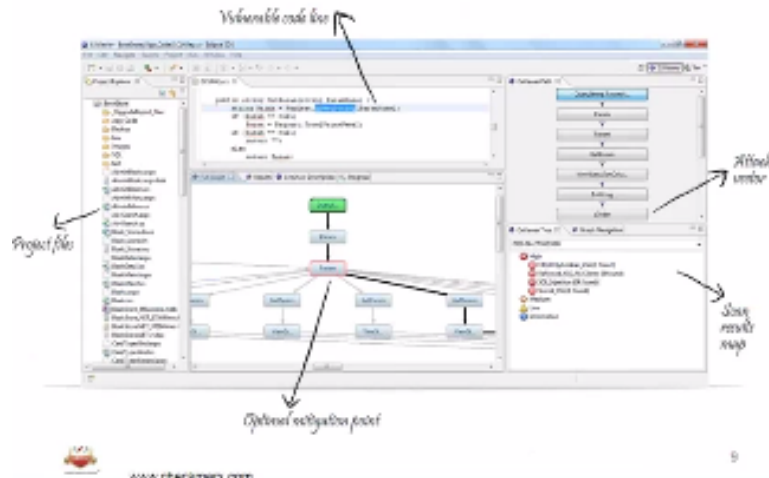
It can be called by vertical people here but here in this kills only it have only one flow of the one direct call used for main delay, so on the reference slide you can see the partial of the how the particular function or the group of program it can highlight, here it is defined in main actually it will list out global as functions as how many functions are there? We have three functions in that tool and it will report particular main and that tricks.

So what is the count line? So what is the count line called? You know that equitable lines of code and defines there and different lines comments and there are inactive there are zero, there are in active lines it will report as so many numbers so likewise you can have the report I will try to show you one of the testing's how we can delete the report with access to and you can also download this tool for draft for commercial use that is www.scitools.com.

They will give a 15 days revaluation machine where you can create sample project sample called embedded C or C++ call project and analyzed it will be same maybe the load part that you can take care maybe I can provide some so we can go through tem and try to understand the tool, so

this is snap shot of the understand for C++ statistic tool, so there is one more tool we can go through.
(Refer Slide Time: 39: 26)

Static testing tools



This is check marks.com you can see eventually this also have a various output in terms of one another all the line what is having a issue? Through here highlight and it restored the project files for that we had to think he is taking example snap shot of I think it is there data based program right hand side you can see flow of different programs it is called attack tester how is being brought out?

Similarly results it will show in right if there is a found error or issues and there is a optical integration part that means how different programs have been controlled and integrated with this. Likewise we can use the static analysis for static analysis report with the help of the check master tool you can test a side and my slide to understanding purpose here you can see a simple program have been done through unit 3programs or the function which are being invoked being considered as complexity.

(Refer Slide Time: 40:57)



That is not the control flow complexity we know that may not be able to see complexities with the extremely tool you can understand the contrary in detail, here you can see the main function calling the shadow of from there is about 6 to 7 main nest level function you can call it has a level 1 the each one of his program or function will call the level 2 individual function there are each one can have 4 or 5 likewise it is the level 2.

At the next of to be first notation is done similarly we have level 3 we have level 4 we have level 5 so it also can report in terms of no issues with a green and issues with of course we have level 6, so this the complexity here we can presidentially say that to level 6 of course difference on each function or its internal tester and you have a different color aspects of the will though flow so it is one of the good example of understand the C++ all to,

So with help of these the user or tester can UN lies how each program will be functioning have been architected? How each function of the functionality or the individual procedures have been used or have been called within the entire embedded systems. So it is very important aspect for static analysis the next important aspect from the static analysis is,
(Refer Slide Time: 43:02)

Static Analysis..

- **WCET: Timing Analysis of Embedded Software**
- Worst Case Execution Time analysis provides the worst possible execution time of the code before using it in a system
- The WCET of a piece of code depends both on the program flow (like loop iterations, decision statements, and function calls), and on architectural factors like caches and pipelines.



WCET it is also called as worst case execution time analysis so we know that timing is very important especially our real time embedded systems, so we need to have a timing aspects clearly analyzed in the program so that we know that what it is going to take? Worst case for a program it could be any functionality in the entire program or it could be a device driver whatever.

So all these have to be analyzed of its time probably the timing that particular piece of software it takes, it takes from two aspects best as well as worst, so both have to be analyzed. So very important aspects in the embedded software system is needs to be analyzed statically, so worst case for the execution time of this worst possible execution time of the core before using it in the system before actually we use the system on the field or the driving testing.

The WSET of a worst case execution time analysis of a piece of code depends on both and the program code like you have for loops, iterations, then we have a decision which are decision statements, it depends o the various function calls that are looking the program, and of course we architectural factors like cache and pipelines, of course embed industry the use cache memory. You know that embedded software will have a cache.

Which is a temporary storage which is being used very frequently and this factors can be honed stood while doing the missed or the worst case time execution, worst case execution time analysis and pipelines, pipeline is an the thing what an important are aspects such your store and executes so these are some of the stages that core processor or the web processor does for executing so for definitely if you use instruction cycle are involved.

Those needs to be analyzed in terms of how many sections it requires, The line of code or a group of line of code or the entire program of the piece of function so very important that have a worst case and these analysis and the embedded software testing so you know that how much it is going to take for worst skills execution of the particular software with the help of the timing analysis we do that and you will report as a area which is going beyond very intended timing as it needs to be fixed by the development thing in terms of optimizing time or reworking on the core whatever it is.

(Refer Slide Time: 46:54)

Static Analysis..

- WCET analysis was used to verify real-time requirements,
- to optimize programs, to compare algorithms and to evaluate hardware
- Among the many measurement
- Tools used are emulators, time-accurate simulators,
- logic analyzers and oscilloscopes, timer readings
- inserted into the software, and software profiling tools



12

So the WCET analysis was used to verify the real time requirements, to optimize programs, to compare algorithms and to evaluate hardware definitely we need to have a comparison of the various modules that are removed formalities module is going to take so that entirely we know that what is more complex what is going to take more and etc. and also we know and analyze the programs which require optimization.

And so and so hardware is behaving in to and it is taking more time to all have 3, 4 drivers I would say a device drivers, each on takes its own time like one time or two many will take another one will take 5 minutes other one takes 10 micro seconds etc so we know that overall the particular program is that is using 3 device ever we will know how much it is going to take, so this two together seconds could be one left operation or right operation.

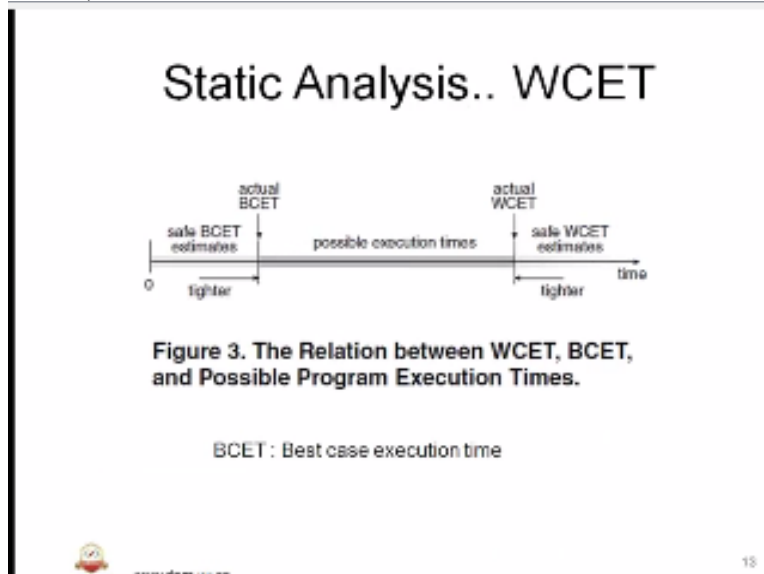
Or access whatever it is so all these can be evaluated with the help of the timing for that particular program, so this also you gave a measurement of the time now be the measurement is timing analysis tools used for emulators, time accurate emulators for the program emulators will support in terms of the consequence and the total time of the calls or the stack usage in terms of how much time it just take or the call tree or the sequence tree etc.

So all these can be done with the help of emulators as well as simulators, when we have logic analyzers which can be looked in to the embedded software and with the help of logic analyzers can be done, and we use oscilloscopes to measure the various timing aspects we know that oscilloscopes helping in the throttle. So we have something like this and we can measure time like this anyways in the oscilloscopes where we have the time on the horizontal axis and we have the value on the vertical axis so that we know what is the value across the time that you going to take.

And we did the analysis with the help of oscilloscopes to probe any signals or the variables, of course we have the timer readings of the particular embedded software procedure or the variable of the signals, and of course we have the inserted programs specifically for measuring these in to the software and we have the profile in the only spend here it software profile. Software profile is the it is a important aspects in the embedded software systems.

With the help of profiling we will do the analysis of the timing that procedure or indented program is going type software, so that is about the worst case mission analysis you can see an example

(Refer Slide Time: 50:53)



This is from normally web reference that are used so her both aspects have been put in this diagram which have BCET and WCET, so BCET is best case execution time and WCET is you know that it is a worst case execution time, so in between in the middle of it you can see that there is a possible execution time and beyond that there is a one say of zone called best case execution analysis.

So the horizontal line you can see the time and the program can take from best to worst so you do a analysis we will use the tighter one and we have a border actually where we define the best and the worst so using this mostly it is going to rescue optimal if it is less than the set it is called as a BCET un estimates of this, if it is beyond that zone of this execution time you can see in the middle it is called as the worst case estimate.

So by this we do an analysis best case execution and the relationship we will draw so as a goal of the WCET analysis is to enter the same and tight estimate of the worst case execution time of the program fragment or the program. So your retaliated problem is that the finding the best case execution time programs you can see the program a example here the figure, how different timing analysis again put together.

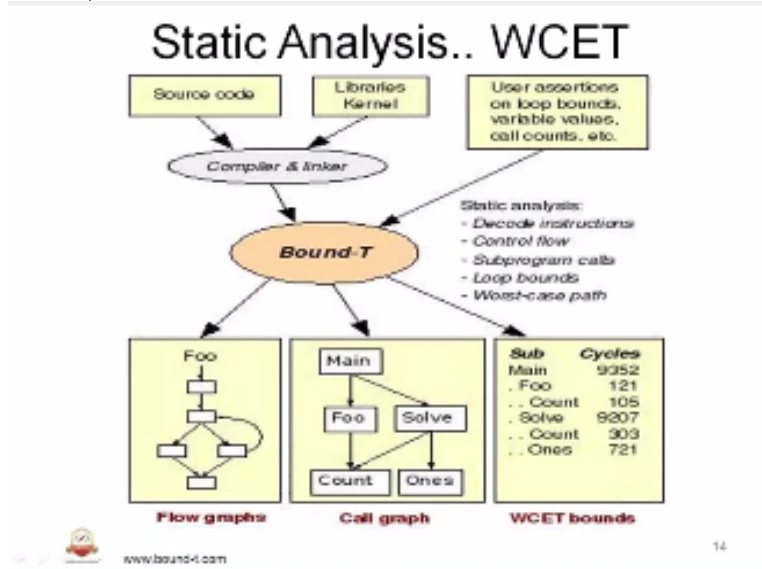
So this can that is the average in terms the best execution time and the tightness safe estimates likewise we can have, so the analytically it is maybe result to arrive ata program execution mode because there are lot of statistical profiling f the input data all are required may it is difficult definitely we may be able to analyze the worst case. And with the intuition and the earliest of various worst case executions we will know.

What will know what the possible execution time and the best execution time? So we for doing that worst case execution analysis so important steps that we took care the program flow I will instruct here, then we need to have low level flow or analysis with the individual low level

analysis will matter in terms of high level program flow collectively and of course we have a calculation of all is flow.

These are very important steps in terms of worst case execution timing analysis, so this is how the worst case timing are done there are other side logic analysis, oscilloscopes, emulators, simulators any of the tools can be used for developing the worst case analysis. You can see another analysis of WCET.

(Refer Slide Time: 55:32)



This is from bounty.com so there is a what is bound to they do means they will define a boundedness of the program, surrounding that they will define the various aspect like source code being used from the compiler and linker we should be applied within the bounty the compiler is linked from tools of various of kernel, and you got assumptions in terms of kernels, values all this will be analyzed in the bounty analysis and the analysis is done with the help of decoding the instructions and control occurs on flow and bounds test link part all will be done with the help of bounty analysis.

The inputs are analyzed and as a result we have flow graph, as a result we have call graph as a result we have oblique's bounty analysis they have test case bounds in terms of the various cycles that each program fragment or the programs will take that main will have a cycles of 9352 cycles. As that program is full which will take one control cycles and the count is 105 it is all 920 and the 303 counts and the ones is 721.

Each program fragment will list out how many cycles? And with help of that we will do a static analysis of the timing of the timing analysis, so the next one is static analysis of stack analysis.

(Refer Slide Time: 57:36)

Static Analysis..

- **Stack analysis:**
- Stack memory has to be allocated statically by the programmer. Underestimating stack usage can lead to serious runtime errors which can be difficult to find. Overestimating stack usage means a waste of memory resources.
- Stack Analysis calculates the stack usage of embedded application.
- The analysis results are valid for all inputs and each task execution.
- Stack analysis can directly analyzed on binary executable, exactly as they are executed in the final system.
- Stack analysis not only reduces development effort but also helps prevent runtime errors due to stack overflow.
- The analysis can also done on the map files generated during the build process.



16

It is also very important aspect in the next type of static analysis is stack analysis you know what is a stack? Stack memory has to be allocated statically by the programmer, that is a implementation code should have a stack memory within the engrafts or of various small functions by bigger functions prevention of the year by smaller functions definitely the program state and the variables are all happening.

Eventually recalled so that is the use of stack definitely embedded systems will have a stack memory and the stack memory has to be allocated how much it is going to take each embedded software fragments or the entire programs underestimating the stack usage can lead to serious runtime errors which are used to have a definitely a state for bound of the stack memory and allocations is very important.

And we should make sure that incrimination of the embedded systems will not go beyond the stack memory so that is why that will be a hard requirement saying that that is 15% of the 78% stack memory should have a that means in worst case it may exceed so that on the safer side better we should reserve 75% of the stack, so that if the program spins beyond 25% should stage because the embedded systems will not crash because there is no stack issue.

So underestimating stack issue can lead to serious run time issues program behavior can be unpredictable when there is a stack issue and this is that is equal to time, similarly was symmetric the also we cannot have a 12 % or 5% buffer allocation because we are wasting the memory, stack analysis calculates the stack usage of embedded application we know that, the analysis results are valid for all inputs and each task execution.

Stack analysis can directly analyzed on binary executable, exactly as they are executed in the final system, stack will be same. Stack analysis not only reduces development effort but also helps prevent runtime error due to stack overflow, so it s a very important aspect is that anytime of the embedded software system program make sure that there is stack overflow during the runtime.

So this also will be analyzed during the stack analysis of the embedded software system in the static analysis, the analysis can also done on the map files generated during the build process, so we know that program will be compiled, linked and as a result we have a executable with map

file basically map file will have a complete picture of the program which is going to be executed on the processor, so the fixed value here it is on the pointers on the memory.

It will list out basically the stack segment memory program test and data, so this needs to be analyzed and definitely this will have a stack memory usage and the upper boundary how much it can have? So that needs to be analyzed and we know with the help of map file what the stack is? So may be as a practical example.

I will try to go through example memory file the memory file will have name with a draft map so this needs to be analyzed statically so with the help of that we know the memory aspects and the stack aspects of the embedded software program, so with this we will conclude this session we will continue the next session on the static analysis with stack overflow and you are all the guide lines that needs to be circumpted for stack analysis in the next session.